Project 2

Coin Picking Robot

Student #	Student Name	% Point	Signature
24042467	Avya Malhotra	143	AM
25550245	Kieran Ross	143	KR
67380998	Nima Karimzadehshirazi	104	NK
99548612	Haoming Gao	70	HG
88299763	William Hui	70	WH
25137316	Yichi Zhang	70	YZ

University of British Columbia
Electrical and Computer Engineering
ELEC 291 - Electrical Engineering Design Studio I
Section 201, L2B winter 2022
Instructor: Dr. Jesus Calvino-Fraga P.Eng.
Date of Submission 8th April

Table of Contents

I. Introduction

For project 2, our given goal was to design, build, and program an autonomous battery-powered coin-picking robot with the ability to individually acquire twenty randomly placed coins across a specified perimeter. The key objectives were to build the robot using any provided microcontrollers (excluding the AT89LP51RC2 or any of its derivatives), incorporate metal detection, electromagnetism, MOSFET drivers, and perimeter detection, all to be completed in C programming.

In terms of hardware, the team initially divided the problem into seven parts **(**refer to Figure 2. *Hardware Flowchart***)**. First, we had to assemble our chosen Microcontroller system, the PIC32MX130, powered and programmed through the BO230XS USB adapter. The second part was creating the electromagnet used to pick the coins up from the ground. Third, was the servo motors that controlled the arm, guiding the electromagnet to the coins. Fourth, was the metal detector using the Colpitts Oscillator. The fifth part was the perimeter detector built with a Tank circuit and the perimeter itself connected to the output of a 555 Timer. The sixth part was the movement of the robot via two 6V operated motors each controlled through an H-bridge. Finally, the last part was the construction of the robot body and the integration of all the different parts onto one breadboard including an optocoupler to reduce noise transfer onto the Microcontroller.

As for software, each part was completed using a function in the base code provided. Our program is sequential, starting at the top every time reset is pressed. The robot initially measures a reference Colpitts Oscillator frequency for when no coin is detected. It then moves straight forward until it detects either a perimeter or a coin. Upon reaching the perimeter, it would back up and rotate 135 degrees counterclockwise before heading straight again. If the coin detector senses a drastic change in frequency, the robot would move back by 10cm and start the servo routine. The servo routine sweeps the electromagnet across in the x-y direction, lifts the coin above the collection box, and turns the electromagnet off to drop the coin. Finally, once the twenty coins have been collected the robot stops until the reset button is pressed **(**refer to Figure 3. *Software Flowchart*).

II. Investigation

a. Idea Generation

Ideas to accomplish project requirements, challenges, and additional features are the foundation of a successful project. Thus, the idea generation stage was revisited constantly throughout the project's timeline. The project's requirements were divided into smaller goals that needed to be accomplished, which would then be compiled into one coherent design. Broadly these divisions included but were not limited to a) Two servo motors, b) H-bridges, c) Metal-detection, d) Perimeter detection, e) Electromagnet, and f) Additional functionality [1]. Ideas for each division were evaluated based on creativity, feasibility, and necessity. Anyone in the group could put forth an idea and anyone could disagree. Ultimately, it was the majority of the group that made the final decision. Most of the project was a direct replication of the examples provided and thus, creativity was not really required. However, for extra functionality, a group member proposed building a custom unique perimeter and utilizing LED lights across the edges for visual appeal; an idea that was adopted once we felt more comfortable with our progress. On the other hand, the idea of identifying the types of coins was suggested but unfortunately not implemented due to a lack of time. Furthermore, idea generation was used to solve bugging errors. For example, when the perimeter detection was not working, it was the abundance of practical ideas pitched by the team that allowed us to fix the issue; as suggested by one of the group members, we first replaced the amplifier, then, as the problem persisted, another idea was pitched that it was one of the inductors causing the issue. Sure enough, with further testing, it was confirmed that the inductor was shorted.

b. Investigation Design

For data gathering, we utilized Project 1, which taught us how to work with frequency oscillations. This helped us with data gathering and experimentation for the coin and perimeter detection portions of this project. Lab #6 also taught us to work with the new Microcontroller systems. In addition, we utilized Piazza as pieces of code were published occasionally. Pulling code from previous labs and Piazza, we documented what each portion of code did and how it affected future code. We left comments and drew flow charts to increase clarity. Our final step was experimenting with our code. Piece by piece, we continued to add to the functionality of our board until we had all the various parts working with each other.

c. Data Collection

This project required lots of precise measurements for optimal functionality. These measurements were needed for areas including coin detection, servo controls, motor controls, and perimeter detection. Our data collection procedure consisted of both qualitative and quantitative analysis. For example, to determine the optimal boundary values for the coin detector, we had to run Max-tests to figure out the largest possible reference oscillations (with no metal present). On the other hand, a qualitative test was required to find the optimal servo-arm routine; we would put a series of angles for both servos to execute in order to determine which combination covered the most area reliably. Furthermore, information such as the project requirements, ideas to implement those requirements, and interesting shortcuts were stored collectively to ensure we effectively met the minimum requirements. Also, through trial and error, our observations were documented on what worked and what needed to be fixed. To store all this information, we used a program called Notion. As a result, we could store information regarding requirements, each rendition of code, and hardware designs and collect additional feature designs in one place. Furthermore, in Notion, we carefully labeled each new section; in return, we could quickly revisit older ideas if need be. In addition, Notion allowed us to upload our updated code for each member to use in real-time.

d. Data Synthesis

Our team synthesized data with VSCode and Notion as we are all able to cooperatively write on VSCode and store any critical sections of code on Notion. To reach subjective conclusions, we typically presented our ideas and voted on them–especially with bonus features and the physical design of the project. Since our group worked in person, we utilized group meetings to finalize our ideas. As for the quantitative tests mentioned above, we used the values to determine optimal function parameters in our code. For the coin detector, if the maximum measured resting frequency was exceeded by more than 5%, we could be sure that metal had been detected. The data collected for the two servos allowed for a smooth coin pick-up and drop-off routine. Qualitative tests done for the motors allowed us to determine the optimal amount by which they would need to reverse and rotate upon coin detection. Finally, the voltage values of the tank circuit for perimeter detection allowed us to confidently establish when the perimeter wire was crossed.

e. Analysis of Results

We came to conclusions regarding the validity of ideas by appraising the difficulty of the idea, how much time we could allot for implementing the idea, and how many people would be required to research and contribute to the idea. Since this was a final project, most of the material for the base requirements had already been taught. However, we used some techniques that required additional research for bonus features. Results regarding tests and data collection were often “good enough” for the robot to complete its task. A relative example where the degrees of error and limitations of theory and measurement came into play was with the coin detector; the maximum recorded resting frequency from 200 measurements would sometimes be surpassed upon a subsequent trial. Therefore, we used a 5% positive deviation to ensure that our boundary would not be exceeded.

III. Design

a. Use of Process

We utilized the seven-step engineering design process. During our first meeting, we defined the problem, listed objectives specifications, and broke the project down into smaller components like pieces of a puzzle. We then set out to identify resources and conduct research. We researched the constraints of the C language and the microcontroller we were utilizing. Next, we listed parts that we needed as a team to implement all the planned functionality. We divided tasks between members based on strength but always allowed members less confident in certain areas to work with those stronger in said areas to allow for novel ideas and collective growth. We then brainstormed possible solutions to our goals, including planning out different sections to the software and hardware requirements. Each section was created and tested for edge cases, bugs, and general improvements. The construction of a complete prototype with basic requirements was our next goal. After several cycles of evaluation and interaction, we arrived at our final product.

b. Need and Constraint Identification

The complete and correct identification of needs and constraints was a vital part of the earlier states of the project. All decisions made would be affected by these constraints and therefore the following were identified in detail.

- A microcontroller-based coin picking robot was to be built by our group. The project was restricted to a microcontroller not part of the 8051 group or any of its derivatives.
- The robot was to be completely autonomous and battery operated. [4]
- The only method of detecting coins allowed was by utilizing a Collpitts oscillator metal detector.
- The robot was to be able to detect all the current Canadian coins in circulation, including but limited to the 0.05$, 0.1$, 0.25$, 1$ & 2$ coins.
- The mechanism to be utilized to pick up each coin was restricted to a manually wound electromagnet.
- All the code for the project had to be completed in C.
- The drive and control of robot motors must utilize MOSFETs (Metal Oxide Semiconductor Field Effect Transistor). The robot must have three degrees of freedom. It should be able to move to any point in two dimensions, and additionally a third degree of freedom as it must be able to control rotation around the z-axis.
- The robot should be able to pick up 20 coins, four of each type, placed randomly in an area defined by a perimeter wire, without leaving this perimeter at any time.
- The robot must be able to detect a perimeter wire carrying an AC current. The minimum area of the perimeter must be 0.5 m^2.

*These needs and constraints were derived from [1]

c. Problem Specification

Several additional design requirements were specified based on needs and constraints.

- Since a frequency generator was initially required to create the AC current in the perimeter wire, and this generator was only available for a limited amount of time per day, we decided to invest time building our own function generator using an oscillator (555 timer) to allow for a functioning perimeter at all times.

- As space was limited, each hardware section was initially built on separate breadboards and then optimized until it fit on one breadboard. This was a requirement set by our team to contribute to the design of the project as a whole.
- In an attempt to allow multiple people to work on coding different hardware sections simultaneously & remotely, we acquired additional PIC32 microcontroller chips so that the team was not restricted to one team member coding and debugging at a time.

d. Solution Generation

The next step in our process was to brainstorm solutions to the newly found problem specifications. We did not condemn any ideas for being unattainable or impractical so that we could allow more raw ideas to enter the conversation.

We immediately saw that the Coin Picking Robot had a need for several moving parts, so servo motors of varying sizes would be essential to the final product. For the ultimate act of picking up coins, one axis of rotation would not be sufficient, so two servo motors would have to be attached to each other with the magnet at one end. The magnet at the end of the servo motor arm would have to be an electromagnet connected to the microcontroller so that it could be toggled on and off in the action of picking up and dropping coins into the receptacle. [3]

Another topic we discussed in our solution generation stage was what perimeter we would use for testing our Coin Picking Robot. If we used a perimeter powered by a function generator it would provide easier and more consistent results; however, we would be locked into only testing in one of the labs. Instead of this, we opted to create our own perimeter using a 555 timer in an A-Stable configuration.

Time permitting, we planned on integrating coin-differentiation code that could detect the coin type based on the frequency change associated with the detection, and this code would keep a running total on the amount of money the robot had accumulated. Also, we talked about the possibility of using a sonic sensor to integrate obstacle detection into our robot. However, all of these additional features had to be disregarded due to the time crunch.

An additional feature that we did successfully integrate was a 3D printed extension onto the front of the robot to reduce noise between the three inductors and increase sensitivity to the perimeters. [4]

e. Solution Evaluation

While mapping out the project timeline, we worked on prototyping parts and testing code blocks individually to see if a given feature would successfully work in our robot.

In testing out minis servo motors paired with the electromagnet, we found that slow and steady movements were the only way to pick up coins and even then we could not pick up the two-dollar coin. This was because our magnet was not strong enough. Possible solutions to this were:

1. Increase the current through the Magnet
2. Increase the resistance of the Magnet by increasing the number of turns of copper wire

We decided to try to intensify the current through the electromagnet instead of having to fully rewire another magnet. Therefore we lowered one of the resistance values in the electromagnet circuit and tried the servo pickup routine again. This time, the coins (from a dime to a toonie) could be picked up with ease and thrown into the coin-carrying receptacle at greater mini servo motor speeds.

To ensure the high quality of our perimeter detecting inductors, we tested them rigorously and independently from the rest of our design. We noticed interference and noise when the three inductors were all lined up at the front, so in order to mitigate this, we tested several possible solutions. First, we replaced some inductors to see if it might be an individual part's flaw. This provided helpful feedback while not solving the problem. We knew it was not the inductors, however when they were close to the metal of the robot the interference was strongest. So secondly, we wrapped the metal in electrical tape and retested it. This only slightly placated the problem. Our final attempt at problem-solving was using a 3D printed robot extension. This proved to be the solution we were looking for, not only solving the problem of interference but also making the perimeter detection even more perceptive.

Eventually, we concluded on our final design due to fierce testing, programmer capabilities, and time constraints. We opted for an efficient one-breadboard design, topped with the sleek 3D printed sensitivity extension.

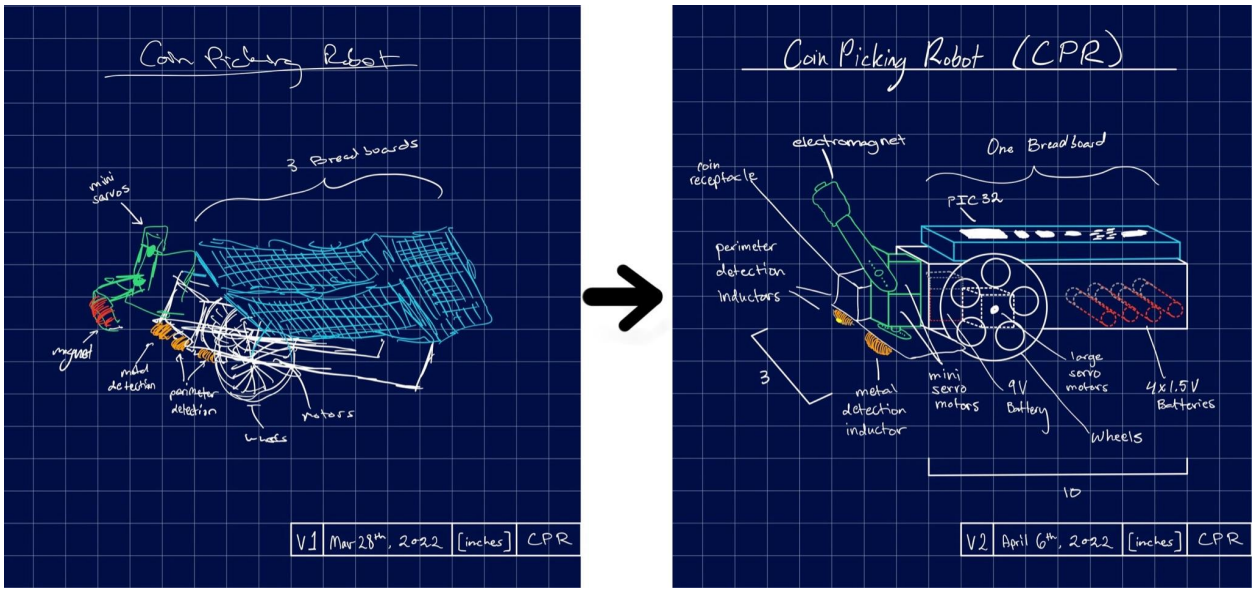


Figure 1. *Design Schematic Evolution*

f. Safety/Professionalism

As we did not use the tools from the MCLD workshop – or any heavy-duty machinery – safety was not an overbearing concern for this project. However, as always, we practiced proper handling of all the equipment that we used. In particular, we wore safety glasses when handling a soldering iron and ensured that the fumes pointed in the direction of the ventilator. Furthermore, we maintained a clean work environment to help prevent loose metals from hurting anyone by accident, while also wearing proper footwear.

Another obvious hazard in any setting is reckless behaviour. Throughout the timeline of the project, our group stayed focused and never used the tools to play around. We also promoted a sense of emotional safety, by treating each other with respect and care. Even if certain team members were not necessarily contributing an equal amount, they were only ever confronted with innocuous yet firm constructive criticisms. A way in which we displayed respect for one another was by being punctual and arriving at planned meetings on time. Finally, each code source used was cited and adequately managed.

g. Detailed Design

Hardware

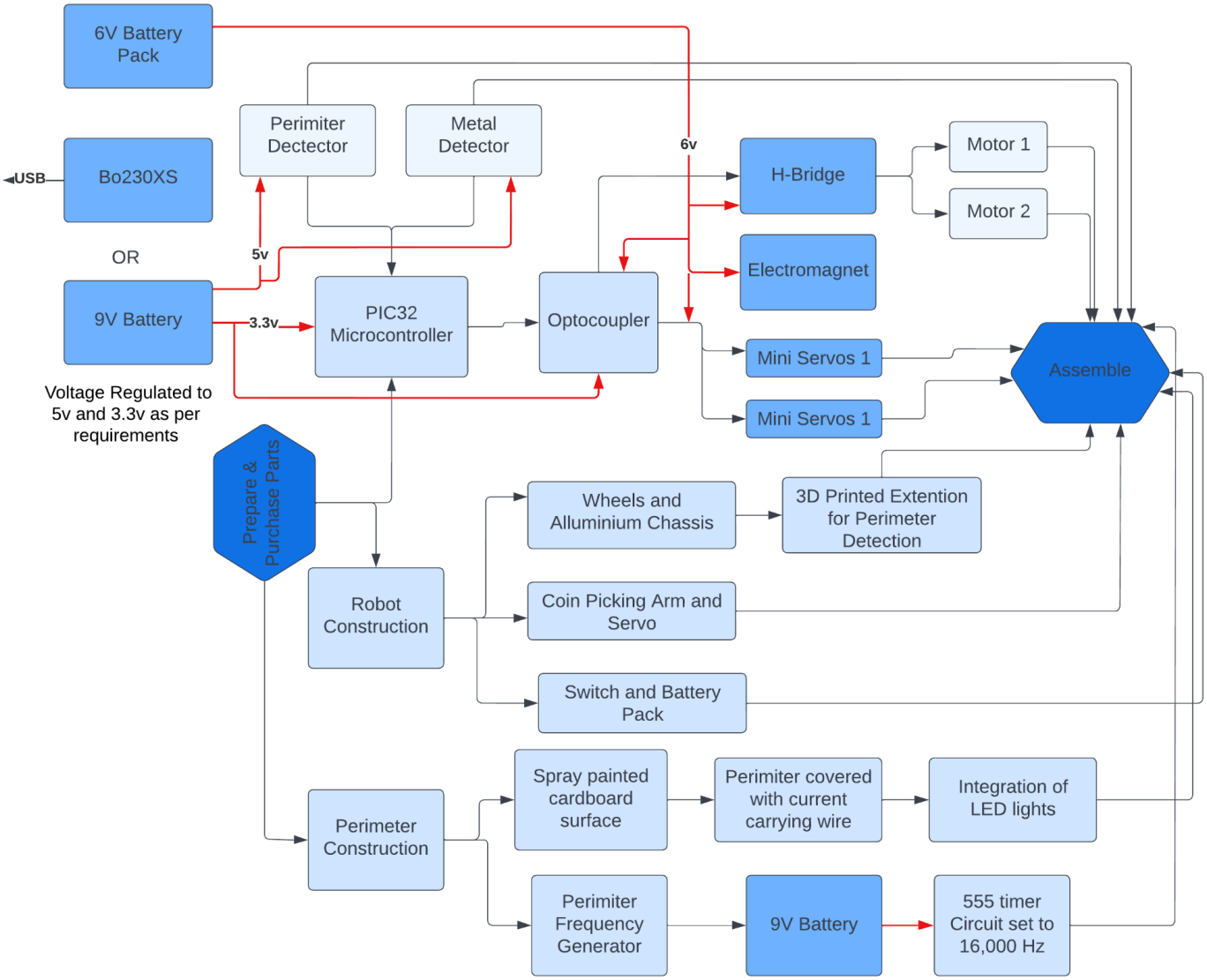


Figure 2. *Hardware Flowchart*

The hardware component was built on many skills we have learned over the past couple of years. For example, building circuits, working with microcontrollers other than the AT89LP51RC2, using servo motors, creating large-scale inductors, and wiring up optocouplers to reduce noise. In addition, we have gained proficiency with hand tools, metal detecting inductors, and constructing a working perimeter circuit of any given frequency using a 555 timer.

PIC32MX130 Microcontroller and BO230XS USB Adapter

The PIC32MX130 microcontroller is one of the five additional microcontrollers we were given to work with for this project, as we were not allowed to use the AT89LP51C2. In Lab 6, our group members all used different microcontrollers. After each member had gained experience with a different system, we discussed all of the various strengths and weaknesses, eventually settling on the PIC32. The PIC32MX130 uses 28 pins with 64k flash and MIPS architecture and is very compatible with coding in C.

In addition, there is the BO230XS USB adapter. As stated in the name, this allows us to connect our breadboard to our operating device. The PIC32MX130 microcontroller allows us to process all commands for our project while the BO230XS connects to our devices. Together, we can program our breadboards as we see fit. The perimeter detection, metal detection, servo motors, electromagnet, and BO230XS USB adapter are all connected to the microcontroller.

Perimeter Detection Circuit

The perimeter detection circuit works with a strong inductor and capacitor in parallel connected to a non-inverting amplifier, the KA358, with the output connected to an MBR150 Schottky diode. The resistance and capacitance values were tweaked so the circuit had a resonant frequency of 16kHz with a gain of 52, thus allowing the voltage across the inductor to both change when approaching a 16kHz square wave circuit (see Perimeter Construction and Generation), but also be amplified to the point of being easily recognizable and consistent.

We connected two of these circuits to our microcontroller, with the two inductors being placed at the front of the robot at a 90-degree angle from each other. The detection works best when the perimeter is approached perpendicular to the orientation of the inductor. Thus with two inductors, the robot would be able to detect the perimeter no matter the angle from which it approached.

Metal Detection Circuit

The metal detection circuit is made with an inductor in parallel with a resistor and CMOS inverter, or an N-FET and a P-FET connected at the gates and drains. When metal is placed near the inductor, there are small changes in inductance which translate into changes in oscillatory frequency. The output of the inverter connects to a microcontroller input, where these diminutive changes can be measured and exemplified.

Servo Motors

Two types of servo motors were implemented. Two larger Solarbotics GM4s would control the wheels of the robot while two mini KY61s would control the Z and θ axes of rotation of the magnet arm.

The Solarbotics GM4s were connected to an H-Bridge and optocouplers, allowing for outputs from the microcontroller to easily send signals to turn the wheels clockwise, counterclockwise or not at all. The H-Bridge consisted of two pairs of N-FETs and P-FETs connected on each side of the inputs of the motors, and the optocouplers we used were the LTV647s. These would reduce the noise in the circuit drastically and without them, fitting everything onto one breadboard would have proved a much more challenging task. The KY61s were also connected to the microcontroller via optocoupler and could be influenced to be any position we wanted through C code.

Electromagnet and Control Circuit

We first constructed the electromagnet by looping copper wire hundreds of times around a base. When tested it measured approximately 9 ohms. The electromagnet control circuit was very straightforward. Output from the microcontroller went through an optocoupler and then the current through the inductor induced a magnetic field, allowing for coin-pickup.

LM7805 Voltage Converter

To allow our robot to run without being connected to a laptop, we needed to add an LM7805 9-volt to 5-volt converter circuit. The 9-volt battery would connect to the input of the LM7805 and ground and then the output would connect to our 5-volt rail, delivering power to the rest of the circuit. We also added 3 stabilizing capacitors to smooth the transition and minimize noise.

Perimeter Construction and Generation

To ensure we would have the flexibility of testing our robot wherever we wanted, we decided to create our own perimeter. We constructed it on a flat cardboard base, about 3x5 feet, spray-painted black, lined with RGB LEDs and a long wire underneath. By connecting a NE555 timer in an A-Stable configuration with the wires in our perimeter, we created a 16kHz square wave all along the rectangular base.

Software

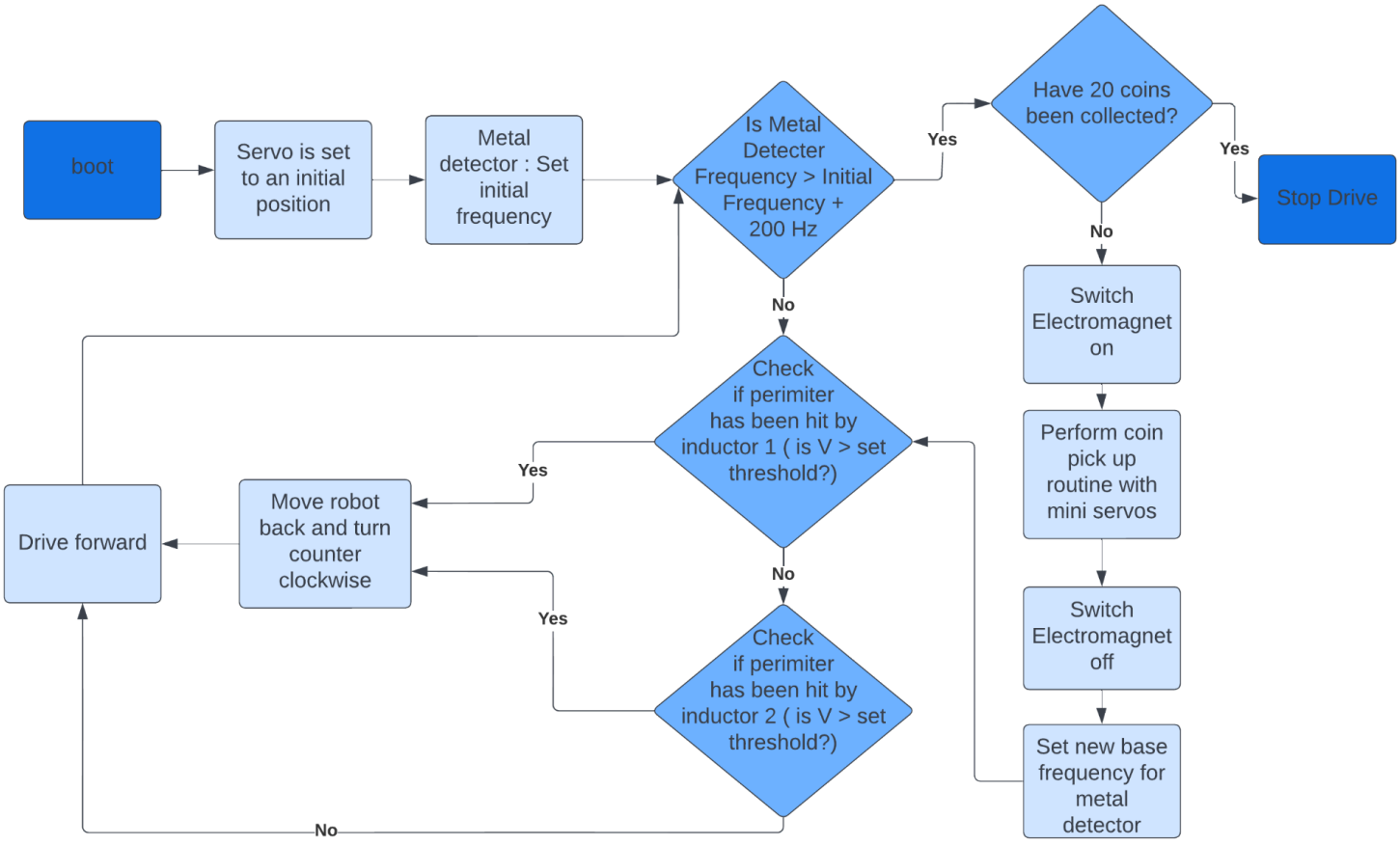


Figure 3. *Software Flowchart*

Robot Drive Control

Drive Control

```
void drive(int directive){
    switch (directive){
        case 0:
            LATAbits.LATA2 = 0;
            LATAbits.LATA0 = 1;
            LATAbits.LATA4 = 0;
            LATBbits.LATB4 = 1;
            break;
        case 1:
            LATAbits.LATA2 = 1;
            LATAbits.LATA0 = 0;
            LATAbits.LATA4 = 1;
            LATBbits.LATB4 = 0;
            break;
        case 2:
            LATAbits.LATA2 = 1;
            LATAbits.LATA0 = 0;
            LATAbits.LATA4 = 0;
            LATBbits.LATB4 = 1;
            break;
        case 3:
            LATAbits.LATA2 = 0;
            LATAbits.LATA0 = 1;
            LATAbits.LATA4 = 1;
            LATBbits.LATB4 = 0;
            break;
        case 4:
            LATAbits.LATA2 = 0;
            LATAbits.LATA0 = 0;
            LATAbits.LATA4 = 0;
            LATBbits.LATB4 = 0;
            break;
        default:
            break;
    }

}
```

The code to the left compliments the H-Bridge Motor Control section of the hardware section. It utilizes four digital output pins of the PIC32, in this case pin A2, A0, A4 & B4 (Corresponding to pin 2, 9, 11 & 12 respectively) each individually set to 1 or 0 to control the motion of servos. There are two servos which are controlled by two output pins each. 1 corresponds to HIGH, whereas 0 corresponds to LOW, and a combination of each leads to varying motions of the servos. As seen, we have decided to create a function with an integer input that can be called in the main() for clean and elegant control of the motion of the robot. Case 0,1,2,3 being forward and backward, clockwise and counter clockwise motion respectively. There is an additional 4th option, in case the robot needs to be able to stand still, such as when it is picking up coins.

Robot Mini Servos Control (Coin picking arm)

```
void servocontrol() {
    int pw;
    drive(1);
    waitms(20);
    drive(4);

    LATAbits.LATA1 = 1;
  // Electromagnet on

    //lower
    for (i = 90; i <= 240; ++i) {
        pw = i;
        ISR_pwm1 = pw;
        waitms(5);
    }

    //sweeping motion
    for (i = 160; i <= 240; ++i) {
        pw = i;
        ISR_pwm2 = pw;
        waitms(5);
    }

    //readjust for raising
    for (i = 240; i >= 170; --i) {
        pw = i;
        ISR_pwm2 = pw;
        waitms(5);
    }

    //raising coin
    for (i = 240; i >= 90; --i) {
        pw = i;
        ISR_pwm1 = pw;
        waitms(5);
    }

    for (i = 170; i >= 120; --i) {
        pw = i;
        ISR_pwm2 = pw;
        waitms(5);
    }

    LATAbits.LATA1 = 0;
  // Electromagent off

    //return to base
    for (i = 120; i <= 160; ++i) {
        pw = i;
        ISR_pwm2 = pw;
        waitms(5);
    }
}
```

Initially we proposed writing a function that would take the current position of the servo, the desired position and the speed with which one would want to move it to that position. However we immediately realized the only use of the servo is a simple routine it must perform once a coin is detected and never again until the next coin. Therefore to maximize efficiency and reduce complexity, we fittingly wrote a function to perform the coin picking routine.

This function is called in the main() if the metal-detection flag is set to 1. We attempted to smooth the servos in many ways, not only to add to the aesthetic with fluid movements, but also to add to the functionality, as smooth movements would allow a strong continuous grip between the electromagnet and coin. We therefore always increment the loop by one number at a time, and utilize a wait function after each iteration instead of increasing the size of incrementation. Changing the variable ISR_pwm 1 & 2 utilize interrupts provided in the base robot code to send the desired PWM signals to the mini servos. Additionally we can set the output pin A1 to high and low. This pin controls the electromagnet, which is turned on and off in this function at the desired servo positions.

Perimeter Detection

Perimeter Detection

```
int perimeterReached(int pin) {
    int threshold = 500;

    int adcval;
    long int v;
    adcval = ADCRead(pin);
    uart_puts("ADC[5]=0x");
    PrintNumber(adcval, 16, 3);
    uart_puts(", V=");
    v = (adcval * 3290L) / 1023L; // 3.290 is VDD
    PrintNumber(v / 1000, 10, 1);
    uart_puts(".");
    PrintNumber(v % 1000, 10, 3);
    uart_puts("V ");

    if (v >= threshold){
        return 1;
    }
    else{
        return 0;
    }
}
```

This piece of code is a testament to the relationship between hardware and software. As the complete and proper working of one simplifies and strengthens the other. Although the perimeter detention is a complex circuit, all the software must do is read the voltage output of two diodes connected to two analog input pins of the PIC32. As in previous labs a simple ADCRead(pin) command is all it takes to retrieve voltage values for both inductors.

The PIC32 has a built-in Analog to Digital Converter and this further simplifies the process [5] . Apart from this both inductors were carefully calibrated to have at least an increase in voltage by 0.5 volts if a wire with a frequency of 16kHz was brought close to it. Therefore a base value and threshold is set checking if the threshold is ever breached, returning a true or false at the end of the function. This function is called and utilized in the main().

Metal Detection

Metal Detection

```
count = GetPeriod(100);

    if (count > 0){

            if (f_new ==0){
                valcompare =((SYSCLK / 2L) * 100L) / count;
            }

            f_new = ((SYSCLK / 2L) * 100L) / count;
            uart_puts("f_new=");
            PrintNumber(f_new, 10, 7);
            uart_puts("  val=");
            PrintNumber(valcompare, 10, 7);
            uart_puts("\n\r");

            if (f_new<(valcompare+200)) {
                //no coin detected
                printf("No coin detected!\n\n\r");
                MetalDetectionFlag = 0;
            }

            else {
                //must stop everything, measure an avg of new
                //frequencies for precice determination of coin type
                printf("Coin detected!\n\n\r");
                //waitms(200);
                MetalDetectionFlag = 1;
            }
        }
   else {
            uart_puts("NO SIGNAL \r");
        }
```

This seemingly inconspicuous piece of code was one that went through the most amount of iterations, as it was challenging to find a balance between complexity and functionality. As we added more components to the robot, switched to just one breadboard, or even soldered the inductor in place, we noticed all minute changes would affect the threshold by impacting noise in our measurements.

Initially we sought to solve this through mathematics, attempting to measure 100 base frequencies, calculating the average to get a better reading. We tested the implementation of a min/max function and even used complex thresholds involving calculating fluctuations. However after eliminating all sources of error on the hardware side, we began getting useful frequency values that were much more reliable. As seen in the code, we retrieve 100 period calculations [7], and calculate the corresponding base frequency. If a new frequency is larger than this base frequency by a given threshold, set a flag which can then be used later on in main(). We also have an else statement for “No signal” mainly utilized for debugging purposes. Last but not least a new

base frequency is calculated each time a coin is picked (As coins collected will inadvertently increase the base frequency measured by the metal detecting inductor). We do this by f_new to 0, so that the if-statement is true after each coin picked.

Solution Assessment

Design Component	Testing Methods and results	Strengths/Weaknesses
Perimeter Detection	Testing for the perimeter detection was mainly done with Putty and reading voltage values while moving the perimeter closer and further away from the inductors. That way, we could tell if they were responsive, or if they were just giving garbage values.	The inductors used for the purpose of perimeter detection were extremely fragile, and replacing them was a hassle as we had soldered these inductors to wires and zip tied them to the base of the robot. Calibration was difficult as several other components would interfere with the perimeter detection, including the frequency of the perimeter wire, and therefore capacitor values had to be changed, and the front end of the robot extended via a custom 3D printed part to increase sensitivity while increasing distance from other circuitry.
Perimeter Creation	Initial testing of the perimeter generator was purely mathematical. A formula was provided in earlier labs that linked the resistor and capacitor values connected to a 555 timer to the frequency it generated. Using this we determined a 500 ohm resistor along with two 100 ohm resistors in series would allow for the required frequency [2]. We confirmed this through utilizing an oscilloscope to measure actual frequency output. We then further tweaked the resistor values as theoretical calculations and real life results were off by a margin of error greater than we initially predicted.	Once set, the perimeter was easy to control and extremely reliable. To prevent a short circuit, a 47 ohm resistor was added in series with the perimeter wire. A noticeable drawback was that this resistor would heat up quite quickly. The perimeter generation was also dependent on the voltage supplied to it, and the circuit would quickly consume 9v batteries, making it a costly affair. We had to utilize 5 volts off of the USB ports of our laptops until the final demo to reduce costs.
Metal Detector	Idea: if the maximum recorded frequency was above the reference by a certain amount,	The capacitors and resistor in the Collpitts circuit were not supposed to

	metal would have to be a contributor which would indicate coin detection. The frequency of the Collpitts oscillator would be measured, first with respect to ground (as the reference), and then with the coin containing the smallest amount of metal (the dime).Thresholds were adjusted constantly as new components including the Tank circuit would generate noise.	output a frequency close to 16kHz as this would cause interference with the Tank circuit used for perimeter detection.
Mini Servo Control (Coin picking arm)	The servos went through several iterations of testing for different factors. Initially they were connected directly into the PIC32 without an optocoupler to ensure the working state of both servos but apply min/max PWM signals of 60 & 240 to determine maximum range of motion [6]. Then a quantitative test was conducted with increasing weights to determine the maximum carrying capacity of each servo. Tests were also conducted to vary the speed of the servo to determine min and mx degree rotations per second. Additionally different voltage levels were tested ranging between 3.3 to 6 to see how this affected performance. Finally a simple servo program was utilized to map all the values both servos would need to move, “the servo routine”, to be able to bring the electromagnet close to a coin, and put it inside the coin box.	Manually finding each position each servo would have to move to was a time consuming process, any change in calibration, such as the time when one servo had to be replaced would result in restarting the process of determining the PWM signals for the coin pick up routine. Additionally the servos would also interfere with the frequency of the metal detector. That said, they were quite simple to program and control, especially parameters such as speed. We were able to get extremely fluid movements using basic C logic. The servos were quite resilient to wear and tear, and forgiving when pin connections to the servos were incorrect.
Motor Control (Wheels)	The H-Bridge controlling the wheels was perhaps the most complex component of hardware design. Utilizing four mosfets and an optocoupler with two different voltages and different grounds, it was a challenge to get it up and running. However, once built, tests were conducted to figure out the combination of High and Low bits for Forward, Backward, Clockwise and Counterclockwise movement. Tests were done evaluating performance of the motors on different surfaces, and different weights on the robot (As 20+ coins and circuitry would	After a while, due to the wheels turning in different directions, the screws became looser/tighter, thus making one wheel more stiff than the other. This caused the robot to swerve to one side while going straight.

	weigh it down). We also switched screws after tests indicated the default screws provided would become loose quite quickly.	
Electromagnet	The main point that needed to be tested is whether the Electromagnet is working. We used different types of coins to test if the electromagnet could pick up coins with heavy enough and light enough weights. When integrating all the parts together, we once found the electromagnet could not pull up coins and checked for the circuit. Using voltmeter, we detected the circuit fragment going short circuit.	While the electromagnet circuit was relatively simple, the act of making the magnet itself strong enough through small tweaks to the circuit's resistance values in order to avoid fully rewiring the inductor was by no means straightforward.

IV. Live-Long Learning

This project not only allowed us to gain a lot of valuable experience but also tested the cooperation and learning ability of the whole team. This project allowed us to explore beyond the 8051 architecture and assess for ourselves under what circumstances one should use the ARM, AVR or the MSP430 family of microcontrollers [2]. The main knowledge gap that the team encountered was the conceptual understanding of the new circuits introduced to us; i.e., the Tank circuit, Collpitts oscillators, H-Bridge and even the optocouplers.

Surprisingly, we overestimated the software component of this project, which our team found quite straightforward. There is no doubt that the complexity of this project comes from the creation of multiple different circuits, and the complete and seamless integration between them. Multiple components have to work in harmony for the robot to be purposeful. Creating a workflow, a time-efficient pipeline of iteration, debugging and implementation was a challenging yet fulfilling task.

A lot of the components in this project were more manageable as they had been covered by previous courses. ELEC 211 taught us about electromagnetism which allowed for a deep understanding of the mechanisms behind a loop of wire with current flowing through it. APSC 101 taught us how to work with PWM signals & servos and Arduino; although we weren't allowed to use Arduino in this project, we used it to quickly diagnose a burnt servo motor!

Finally, computer programming courses, most prominently CPSC 259, equipped us with the ability to quickly and reliably debug programming issues and translate real-world problems into code. The logic of testbenches taught in CPEN 211 was applied to integrate multiple circuits piece by piece.

This project has also been a catalyst for many in the team in terms of finding what part of electrical engineering they wish to pursue further, be it circuitry, firmware, or software. It has taught us how to strike a balance between aesthetics and functionality, and through unremitting efforts and perseverance, we have successfully made the robot work. Our team members believe that the knowledge and experience will be invaluable to us as electrical engineering students and to our future internships and work.

V. Conclusion

In this project, our objective was to make a robot cart that could detect and collect coins at the same time. Group members first grouped and made plans according to the project literature, then completed their respective tasks according to the content of the plan and sought help from peers and professors when they encountered difficulties.

In order to complete the project, we were required to create a metal detection system, coin picking mechanism and electromagnetic system, all while not using the AT89LP51C2 microcontroller. Additionally, all programming for the project was constrained to C code.

During our design process, we divided into two groups to design hardware and software separately. Members would complete their assignments at home and on reassembling as a team, we would integrate them together. The biggest challenge the team encountered was the final assembly of the robot. There were some circuits that had no problems in independent testing but had problems during assembly, causing the board to smoke. Later, with the continuous efforts and tests of the two members, the circuit connection where the problem was located was successfully solved.

Overall, the project took us about 80 hours to complete, a lot of that time was spent fixing bugs in the circuit and editing the code. This project has been a great way for our team to learn about circuit connection, code wiring, and assembling hardware. Moreover, we learned how to cooperate, help each other and overcome difficult problems while working as a team.

VI. References

[1] Calvino-Fraga, Jesus, “ELEC291_Project_2_Coin_Picking_Robot_2022.pdf”, University of British Columbia, Vancouver, 2022.

[2] Calvino-Fraga, Jesus, “Lecture slides. ELEC291_Project_2_2022.pdf”, University of British Columbia, Vancouver, 2020.

[3] Calvino-Fraga, Jesus, “Coin_picker_assembly.pdf”, University of British Columbia, Vancouver, 2022.

[4] Calvino-Fraga, Jesus, 2022, “Robot_assembly.pdf”, University of British Columbia, Vancouver.

[5] Calvino-Fraga, Jesus, 2022, “The PIC32 Microcontroller System.pdf”, University of British Columbia, Vancouver.

[6] Calvino-Fraga, Jesus, 2022, PIC32_Robot_Base_Code.zip, University of British Columbia, Vancouver.

[7] Calvino-Fraga, Jesus, 2022, PIC32.zip, University of British Columbia, Vancouver.

VII. Bibliography

Microchip Technology, “PIC32MX Family Datasheet”, PIC32MX130 IC datasheet, 2008.

Fair Child Semiconductor. “KA358 dual operational amplifier”, KA258/KA258A, KA358/KA358A, KA2904 Datasheet, 2011.

National Semiconductor, “LM357 MOSFET”, N-type Datasheet, 2004.

ON Semiconductor, “NTD2955, NVD2955”, MOSFET – Power P-Channel, DPAK, 2019.

Texas Instruments, “LM555 Timer”, LM555 datasheet, 2015.

CUI Devices, “CEM-1203”, Magnetic Buzzer Transducer specification, 2019. Figure

VIII. Appendix

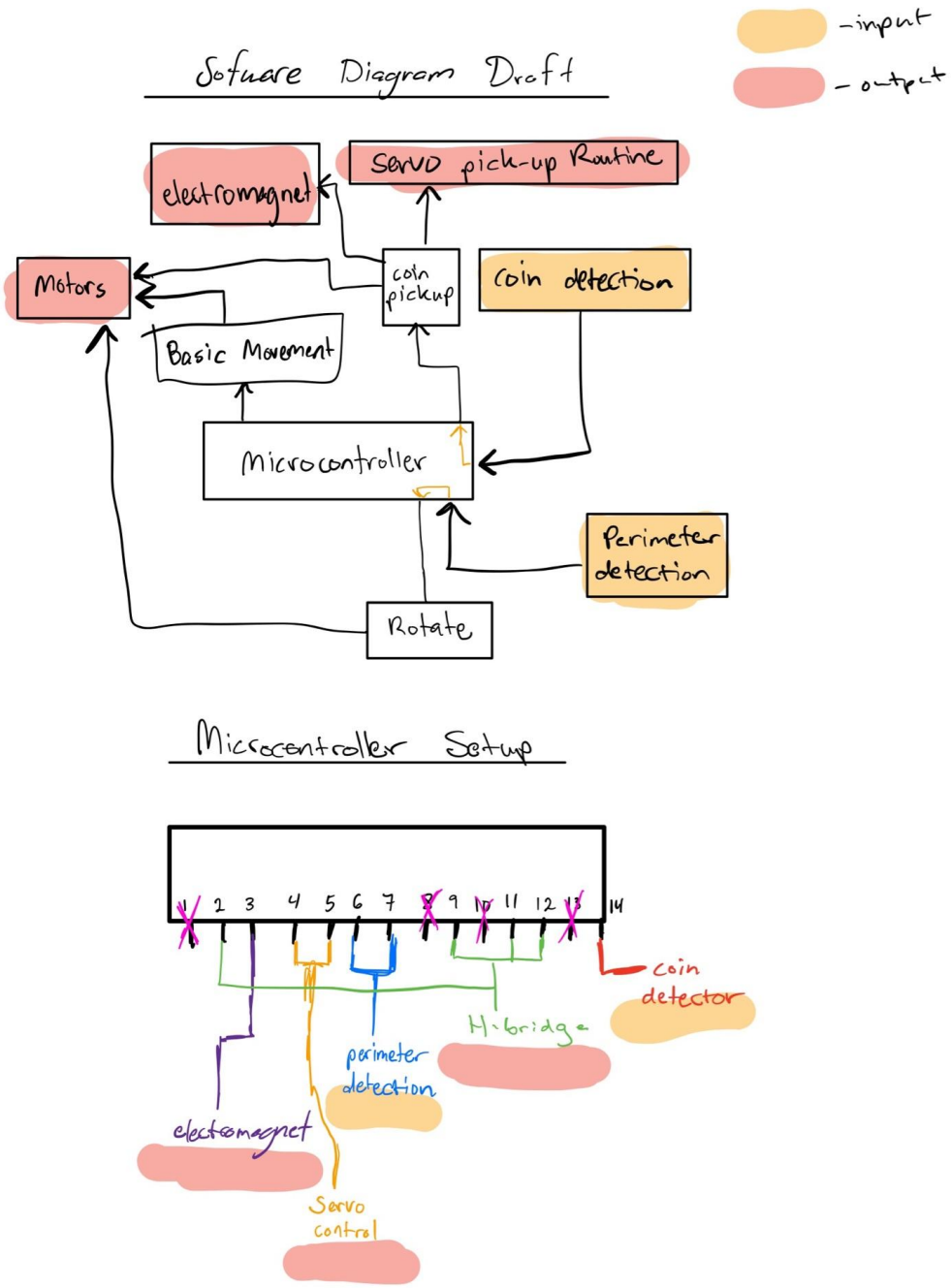
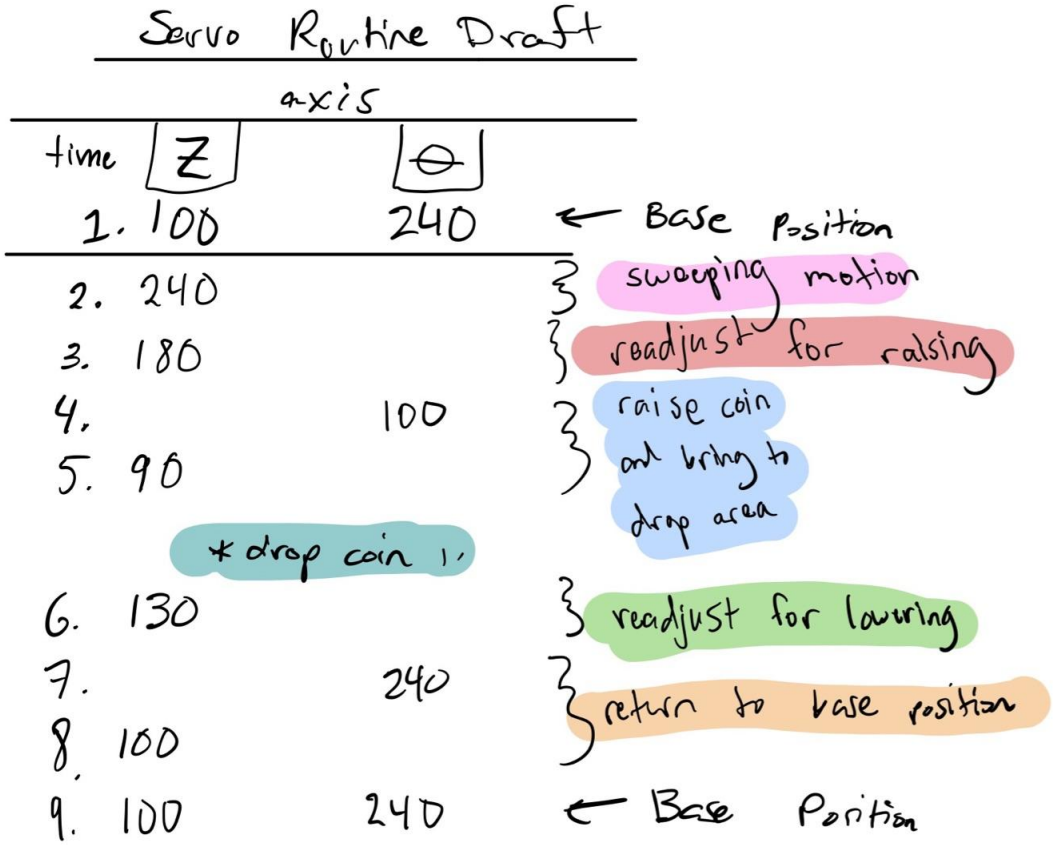


Figure 4. *Software Diagram and Microcontroller Setup*



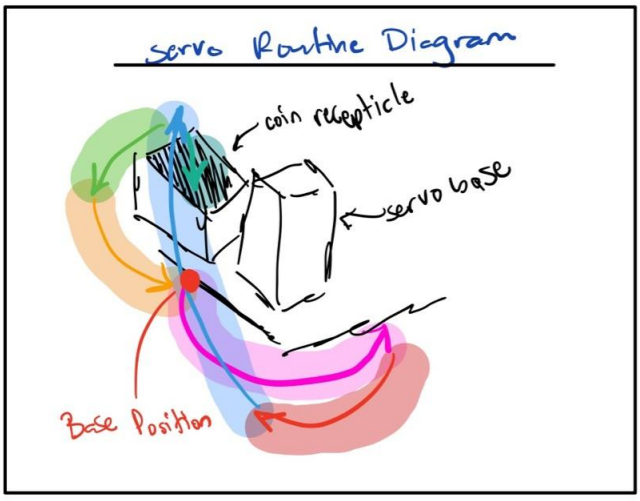


Figure 5. *Servo Routine Draft*

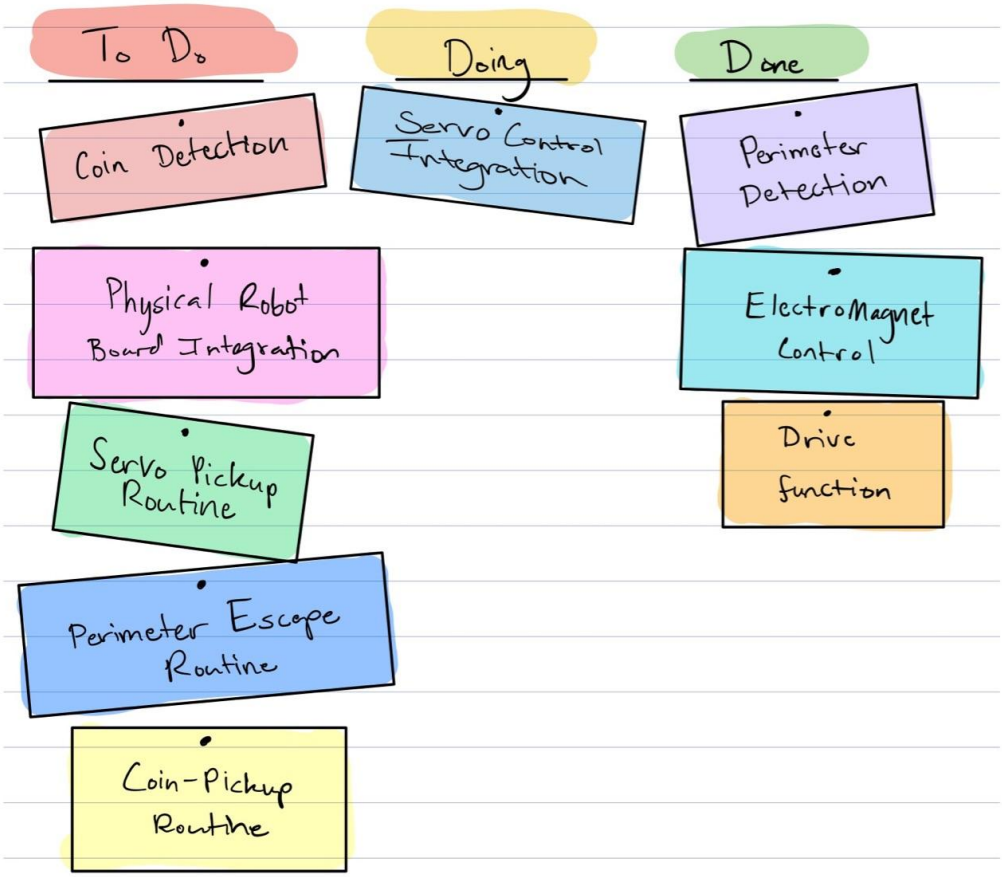


Figure 6. *Working To-Do List - How our group kept organized*